VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Offline vyhledávání adres z předzpracovaných dat projektu OpenStreetMap pro platformu Android

Offline Android-based Address Search on Pre-processed OpenStreetMap Data

2013 Šimon Dorociak

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Šimon Dorociak**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Offline vyhledávání adres z předzpracovaných dat projektu OpenStreetMap pro platformu Android
Offline Android-based Address Search on Pre-processed OpenStreetMap Data

Zásady pro vypracování:

Cílem práce je vytvořit (nebo zvolit existující) datový formát pro uložení adresních dat, exportovaných z projektu OpenStreetMap a umožnit rychlé vyhledávání v databázi těchto bodů. Použité rozhraní bude nabízet alternativní implementaci k android.location.Geocoder. Výstupem bude platformově nezávislý konverzní nástroj pro desktop (bude-li zapotřebí), knihovna tříd realizující požadovanou funkčnost, vzorová aplikace a výsledek srovnání testů vytvořené komponenty s on-line přístupem.

1. Vyhledejte případné konkurenční aplikace či knihovny, které poskytují podobnou funkčnost a srovnejte jejich funkčnost.
2. Zdokumentujte formát, v němž budou data ukládána. Vyberte, které části Android API využijete a v jaké verzi. Popište případné nestandardní knihovny a nástroje, které budete využívat.
3. Popište algoritmy, použité pro rychlé vyhledávání v adresních datech.
4. Analyzujte, navrhněte a implementujte knihovnu pro platformu Android spolu s mobilní aplikací, demonstrující její funkčnost a exportní modul dat z OpenStreetMap pro PC (bude-li to potřeba).
5. Výsledné řešení otestujte alespoň na dvou různých mobilních zařízeních a zhodnoťte dosažené výsledky. Srovnejte funkčnost s referenčním API.

Seznam doporučené odborné literatury:

[1] Burnette, E.: Hello, Android: Introducing Google's Mobile Development Platform. Pragmatic Bookshelf, 2008. ISBN: 978-1-93435-617-3.
[2] Meier, R.: Professional Android 2 Application Development. Wrox Press, 2010. ISBN: 0-47056-552-0.
[3] Projekt OpenStreetMap [online]. [cit. 2012-08-12]. Dostupné z: http://www.openstreetmap.org/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. apríla 2013 .....

Rád by som na tomto mieste poďakoval pánovi Ing. Pavlovi Moravcovi, Ph.D. za odbornú pomoc, veľmi cenné rady, konzultácie a čas, ktorý si na mňa našiel pri vytváraní tejto bakalárskej práce.

Abstrakt

Cieľom tejto práce bolo implementovať knižnicu a ukážkovú aplikáciu pre operačný systém Android, ktorá bude slúžiť na offline vyhľadávanie adries exportovaných z open-source projektu OpenStreetMap. Knižnica predstavuje alternatívnu implementáciu k existujúcemu API android.location.Geocoder slúžiacemu na klasické a reverzné geokódovanie, ktoré však neposkytuje prácu offline. Knižnica za účelom rýchleho vyhľadávania používa SQLite databázu používanú pri klasickom geokódovaní a stromovú štruktúru používanú pri reverznom geokódovaní. Keďže adresné dáta sú dvojrozmerné bol použitý na ich perzistenciu kvadrantový strom, ktorý je primárne navrhnutý na delenie dvojrozmerného priestoru a reprezentáciu dvojrozmerných bodových dat.

Klíčová slova: Android, SQLite, OpenStreetMap, Geokóding, Kvadrantový strom

Abstract

The goal of this thesis has been the implementation of a library and a sample mobile application for Android operating system, which will be used for offline address search on data exported from the open-source OpenStreetMap project. The library represents an alternative implementation to existing android.location.Geocoder API which serves both for the forward and reverse geocoding but doesn't provide offline search. The developed library uses SQLite database for forward geocoding and a tree-based data structure for reverse geocoding, both to obtain as fast search results as possible. Since address data is two-dimensional, we have used the quad tree – which is primary designated to partition two-dimensional space and to represent two-dimensional points – for their persistence.

Keywords: Android, SQLite, OpenStreetMap, Geocoding, Quad tree

Seznam použitých zkratek a symbolů

OSM	–	OpenStreetMap
XML	–	Extensible Markup Language
UI	–	User Interface
DEX	–	Dalvik Executable File
SDK	–	Software Development Kit
NDK	–	Native Development Kit
aapt	–	Android Asset Packaging Tool
API	–	Application Programming Interface
ADT	–	Android Development Tools
AVD	–	Android Virtual Devices
adb	–	Android Debug Bridge
ddms	–	Dalvik Debug Monitor Server
BST	–	Binary Search Tree
DAO	–	Data Access Object
SQL	–	Structured Query Language
K-NN	–	K-Nearest Neighbor algorithm

Obsah

Zoznam tabuliek

Zoznam obrázkov

Seznam výpisů zdrojového kódu

1 Úvod

Posledné 3–4 roky môžme charakterizovať z hľadiska vývoja nových zariadení ako éru chytrých telefónov. Tieto telefóny potrebovali podporu kvalitného operačného systému. Jedným z nich je Android.

Android je mobilný operačný systém založený na modifikovanej verzii Linuxu. Ovládol trh so smartfónmi a zároveň sa stal obstojným konkurentom pre iOS od spoločnosti Apple. V roku 2005 ho kúpila spoločnosť Google a začala vývoj. Google chcel, aby bol systém *open–source* a zadarmo a preto je väčšina kódu uvoľnená pod licenciou *open-source Apache Licence* čo znamená, že ten, kto chce používať Android si môže stiahnuť plný zdrojový kód [1].

Navyše predajcovia (väčšinou výrobcovia hardvéru) môžu vkladať svoje vlastné rozšírenia a prispôsobiť systém s cieľom odlíšiť ich produkty od iných. Tento jednoduchý vývoj robí Android veľmi atraktívnym a tak vzbudil záujem mnohých predajcov. To bolo obzvlášť typické pre spoločnosti ovplyvnené fenoménom *iPhonu* spoločnosti Apple, ktorý odštartoval revolúciu na trhu so smartfónmi [1].

Hlavná výhoda platformy Android je, že ponúka jednotný prístup k vývoju aplikácií. Stačí, že vývojári vyvíjajú pre Android a ich aplikácie sú schopné fungovať na veľkom počte rôznych zariadení s platformou Android. Vo svete smartfónov sú práve aplikácie tou najdôležitejšou časťou úspechu. Výrobcovia zariadení preto vidia Android ako najväčšiu nádej schopnú vzdorovať náporu spoločnosti Apple [1].

Cieľom tejto bakalárskej práce je vytvoriť knižnicu a ukážkovú aplikáciu pre operačný systém Android, ktorá slúži na **offline** vyhľadávanie adries z pred–spracovaných adresných dat exportovaných z *open–source* projektu OpenStreetMap. Je vhodné spomenúť, že podobnú funkčnosť ponúka aplikácia Osmosis avšak tá je určená len pre desktop.
Najdôležitejšou časťou aplikácie je spôsob uloženia adresných dat. Bolo potrebné zvoliť alebo vytvoriť vhodný dátový formát pre uloženie adresných dat, ktorý umožní ich rýchle vyhľadávanie. Finálne budú tieto dáta uložené na pamäťovej karte mobilného zariadenia vo forme binárnych súborov v predvolených priečinkoch.

Samotná aplikácia funguje tak, že spracuje užívateľský vstup, ktorým môže byť čiastočná hľadaná adresa alebo jej približné súradnice, spustí vyhľadávanie a vráti užívateľovi špecifické výsledky, ktorými môžu byť dodatočné informácie o adrese (mesto, PSČ,...) ak zadal čiastočnú adresu alebo niekoľko najlepších výsledkov (adries) ak zadal jej približné súradnice.

Pri vyhľadávaní adries aplikácia používa `SQLite` databázu operačného systému Android a stromovú štruktúru realizovanú kvadrantovým stromom.

Následujúci text môžme rozdeliť na niekoľko častí. V prvej časti textu načrtnem základné informácie o operačnom systéme Android, jeho históriu a architektúru a podám stručné informácie o *open–source* projekte OpenStreetMap a stromových štruktúrach.

V druhej časti sa budem zaoberať samotným vývojom aplikácie a v poslednej zhodnotím dosiahnuté výsledky a zhrniem celú prácu vrátane výhľadov do budúcnosti.

2 Android

2.1 Všeobecná charakteristika

Jedná sa o *open-source* platformu určenú predovšetkým pre smartfóny. OS beží na linuxovom jadre, ktoré zaisťuje správu pamäte, procesorov, siete, ovládačov, prístup k senzorom atď. [3].

Keďže sa jedná o systém pre mobilné zariadenia, vývojári museli celý systém prispôsobiť pre hardvér s nižšou výkonnosťou preto je aj základný jazyk *Android Dalvik VM 1* optimalizovaný pre výkonnostne nižší hardvér [3]. Pre využitie hlavného potenciálu Android aplikácií ako sú napríklad: *mapy, instant messaging, youtube alebo google browser* je nutné byť pripojený k Internetu.

Oficiálne doporučené a zároveň podporované vývojové prostredie pre Android je **Eclipse** ale aplikácie je možné kompilovať aj cez **príkazový riadok** alebo je možné použiť aj iné vývojové prostredie ako napr. **NetBeans**.

2.2 História

História siaha až do októbra roku 2003 kedy bola v Kalifornii založená spoločnosť **Android Inc**. V roku 2005 spoločnosť odkúpil Google. Následne 5.11. 2007 spoločnosť vydala prvú ukážkovú verziu `SDK` Androidu, ktorá obsahovala [2, 3]:

- Vývojové prostredie
- Emulátor
- Debugger
- Ukážkové programy

V tom istom roku bolo založené konzorcium **Open Handset Alliance (OHA)**, ktorého cieľom je vytvárať a presadzovať štandardy vývoja softvéru a hardvéru pre mobilné zariadenia. Dnes už takmer všetci výrobcovia mobilných telefónov spolupracujú s OHA. Výnimkou sú spoločnosti Nokia a Apple. Dňa 23.9. 2008 bol oficiálne predstavený Android 1.0 s vývojovým prostredím [3].

2.3 Architektúra

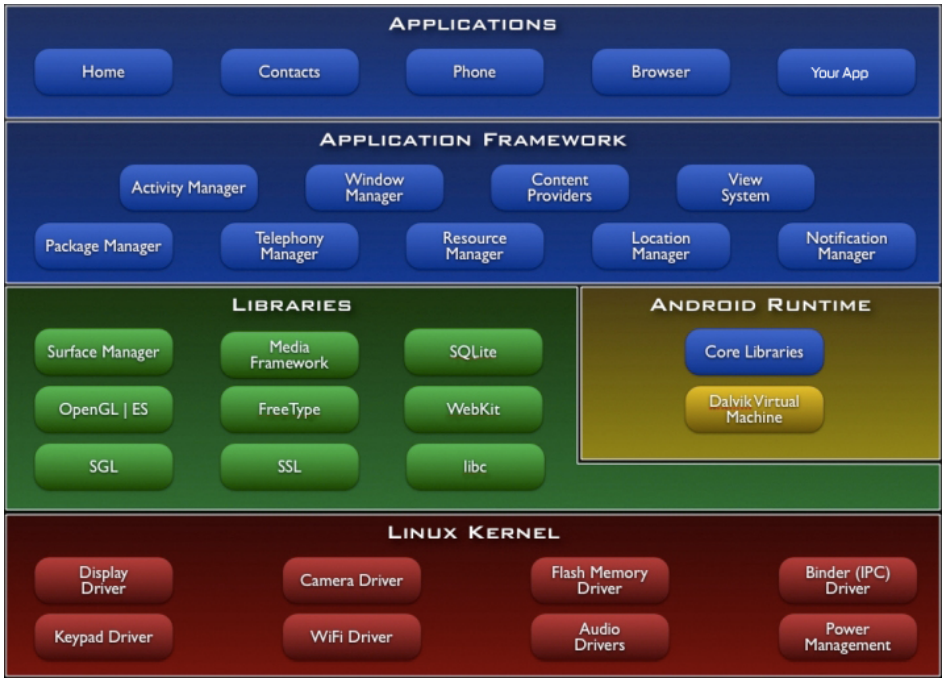
Platforma sa okrem linuxového jadra skladá z množstva knižníc napísaných v jazykoch C/C++ ako napríklad: systémová *libc*, knižnice pre prácu s médiami, knižnice enginu webového browsera, grafického enginu, SQLite a mnoho ďalších [2]. Celá architektúra systému Android je na obrázku 1.

Systém ďalej obsahuje **Dalvik Virtual Machine**, ktorý slúži na vykonávanie bytecodu avšak používaný bytecode není Java bytecode ale Dalvik bytecode.

V štandardnej Jave je zdrojový kód skompilovaný do Java bytecode, ktorý je uložený v .class súboroch. Každá trieda napísaná v zdrojovom kóde bude mať vlastný .class súbor. To znamená, že ak máme jeden .java zdrojový súbor, ktorý obsahuje jednu verejnú triedu, ktorá má vlastnú statickú vnorenú triedu a 3 anonymné triedy, výstupom bude 5 .class súborov. Na platforme Android tomu tak úplne nie je. Aj tu rovnako ako v štandardnej Jave je zdrojový kód skompilovaný do .class súborov. Avšak po vygenerovaní .class súborov sú tieto súbory skonvertované do .dex súboru alebo taktiež **Dalvik Executable File** pomocou **dx** nástroja, ktorý je súčasťou Android SDK [7].

Zatiaľ čo .class súbor obsahuje len jednu triedu, .dex súbor obsahuje niekoľko tried. Je to práve .dex súbor, ktorý je spustený na Dalvik Virtual Machine a ktorý je zároveň optimalizovaný pre mobilné zariadenia z hľadiska využitia pamäte. Pri danom konvertovaní sa taktiež konvertuje Java bytecode do Dalvik bytecode [7].

Avšak je vhodné poznamenať, že prevod není stopercentný. Nie všetko z čistej Javy je možné použiť rovnako aj v Androide teda aplikácie sa neprogramujú v plnohodnotnej Jave [2].



Obr. 1: Diagram architektúry systému Android [3]

2.3.1 SQLite databáza

Jedná sa o embedded `SQL` databázu kde kompletná databáza spolu so všetkými tabuľkami, triggermi a pohľadmi je uložená v jednom súbore. Rovnako sa jedná aj o kompaktnú knižnicu, ktorej veľkosť môže byť s plnou funkčnosťou menšia ako **350 KiB**. Samozrejme toto môže byť ovplyvnené najmä platformou a rôznymi optimalizačnými nastaveniami. [4].

`SQLite` je jednou z najlepších volieb týkajúcich sa výberu databázy pre zariadenia z omedzenou kapacitou pamäti (smartfóny, PDA,...) pretože má veľmi nízke nároky z hľadiska využitia pamäte (minimálne využitie priestoru v stacku (cca **4 KiB**), veľmi malá heap (cca **100 KiB**)) a čo je najhlavnejšie výstupom je databázový formát, ktorý je **platformovo nezávislý** [4].

2.3.2 Logcat

`Logcat` je logovací systém, ktorý poskytuje mechanizmus pre zhromažďovanie a prezeranie systémom ladených výstupov [5].

2.4 Vývoj aplikácií pre platformu Android

Nato, aby sme boli schopní začať s vývojom aplikácií pre platformu Android je nutné mať nainštalovaný Android `SDK`. Je dostupné pre všetky hlavné platformy: Linux, Mac aj Windows. Pre vývoj aplikácií na ktoré sú kladené vysoké požiadavky z hľadiska rýchlosti(prevažne sa jedná o hry) je možné použiť Android **NDK (Native Development Kit)**, ktorý predstavuje balík pre vývoj aplikácií v `C/C++`.

Okrem `SDK` a `NDK` je možné si stiahnuť **ADT plugin** pre Eclipse, ktorý značne uľahčuje vývoj a ladenie Android aplikácií [2].

2.4.1 Android SDK a jeho obsah

Samotné SDK obsahuje niekoľko programov. Určite medzi tie najdôležitejšie patrí program `android`, ktorým sa spúšťa Android SDK a AVD Manager cez ktorý sa sťahujú jednotlivé komponenty SDK ako rôzne verzie Androidu a ďalšie.

Ďalšou dôležitou komponentou je **emulátor**. Emulátor je samostatná desktopová aplikácia, ktorá umožňuje testovať mobilné aplikácie priamo na počítači bez nutnosti ich inštalácie na reálnom mobilnom zariadení. Pomocou vyššie spomenutého AVD Manageru je možné vytvárať a spúšťať nakonfigurované inštancie emulátoru (Android Virtual Devices) [2].

Týmto spôsobom je možné vymodelovať veľmi presne vlastnosti konkrétneho zariadenia. Pre spustenie inštancií emulátoru je možné taktiež použiť príkaz `emulator`, ktorý má širokú škálu parametrov slúžiacich k ďalším úpravám konfigurácie AVD.

Avšak stále emulátor **nie je** reálne mobilné zariadenie napriek širokým možnostiam jeho konfigurácie a preto nesie so sebou isté **obmedzenia** medzi ktoré patria [5]:

- podpora Bluetooth
- detekcia vysunutia alebo zasunutia SD karty
- určenie stavu pripojenej siete, batérie a stavu jej nabitia
- chýba taktiež podpora slúchadiel
- volanie a prijímanie hovorov (možná je len ich simulácia prostredníctvom konzoly emulátora)

Za zmienku stojí ešte spomenúť program `ddms`, ktorý spúšťa Dalvik Debug Monitor – nástroj umožňujúci ladenie aplikácií. Program dokáže komunikovať s aktuálne bežiacimi inštanciami emulátorov ako aj s pripojenými zariadeniami. Pre ladenie aplikácií priamo v telefóne je nutné povoliť v telefóne ladenie cez USB [2].

3 Projekt OpenStreetMap

3.1 O projekte

OpenStreetMap je *open–source* projekt, ktorého cieľom je tvorba geografických dát ako napríklad cestné mapy. Dáta používa najmä s *GPS* prijímačov v režime automatického zaznamenania súradníc prechádzanej trasy, ktoré sú následne kontrolované a upravované. Rozhranie mapovania zahŕňa celý svet, hlavné úsilie sa avšak odohráva vo Francúzsku, na ostrove Man, v Škandinávii a v Spojenom kráľovstve. Pokrytie USA sa docielilo použitím dát Tiger. Na Slovensku a v Čechách je k dispozícii katastrálny portál [6].

Aktuálny stav o importe adresných bodov ČR je k dispozícii na `http://wiki.openstreetmap.org/wiki/Import_Adres_%C4%8CR`.

Mapy sú 2–rozmerné, nezobrazuje sa v nich nadmorská výška a vrstevnice. Vrstevnice treba aplikovať z externého zdroja [6].

Projekt bol založený v júli 2004. Zakladateľom projektu je **Steve Coast** z Veľkej Británie.

3.2 Formát

Hlavným formátom je `XML`, ktorý používa väčšina nástrojov pracujúcich s `OSM` dátami. Je to v podstate zoznam inštancií základných dátových štruktúr medzi ktoré patrí: **Uzol**, **Cesta** a **Relácia** [6]. Nás budú zaujímať výhradne uzly ktoré poskytujú potrebné informácie.

Výhody:

- Dobrá čitateľnosť vďaka zrozumiteľnej a jasnej štruktúre
- Môže byť 100% `ASCII` text
- Nezávislý na programovacom jazyku
- Parsovateľný pomocou rôznych `XML` parserov

Nevýhody:

- Parsovanie môže trvať aj niekoľko desiatok minút (v závislosti na veľkosti dátového zdroja)
- Súbory dosahujú veľmi obrovských veľkostí

3.2.1 Ukážka formátu

Následujúci výpis 1 je ukážkou zjednodušeného avšak kompletného `OSM` súboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap␣0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.090174000" lon="12.248263200" user="SvenHRO" uid="46882"
       visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.090630900" lon="12.244192400" user="PikoWinter" uid="36744"
       visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381"
       user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">
  <tag k="name" v="Neu␣Broderstorf"/>
  <tag k=" traffic_sign "  v=" city_limit "/>
 </node>
  ...
 <node id="298884272" lat="54.09014400" lon="12.251651300" user="SvenHRO" uid="46882"
       visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606"
       timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
   ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower␣Strasse"/>
 </way>
 <relation  id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637"
        timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
   ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
   ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Kustenbus␣Linie␣123"/>
 </relation>
  ...
</osm>
```

Výpis 1: OSM XML formát [6]

4 Stromové štruktúry

4.1 Stromy

Stromy sú jedny z najsilnejších a najefektívnejších pokročilých dátových štruktúr, ktoré simulujú hierarchickú stromovú štruktúru pomocou prepojených uzlov [8, 9].

Strom môžme definovať formálne ako konečný súbor T jedného alebo viacerých uzlov kde [10]:

- Existuje jeden špeciálny uzol nazývaný koreň stromu
- Zostávajúce uzly(s výnimkou koreňového) sú rozdelené do $m \geq 0$ disjunktných množín $T_1, ..., T_m$ kde každá z nich reprezentuje opäť strom. Stromy $T_1, ..., T_m$ sa nazývajú pod-stromami koreňa.

Zjednodušený príklad stromovej štruktúry môžme vidieť na obrázku 2.

Uzol

Uzol je dátová štruktúra [8], ktorá môže obsahovať hodnotu, stav alebo nejakú d'alšiu štruktúru. Každý uzol môže mať najviac jedného rodiča a môže mať nula až n potomkov, ktorí sú o úroveň nižšie (stromy podľa konvencie rastú smerom dole) ako ich rodič pričom:

- Uzol, ktorý nemá žiadneho potomka sa nazýva **list** (koncový uzol)
- Uzol, ktorý má aspoň jedného potomka je **nadradeným uzlom** (predkom) potomka

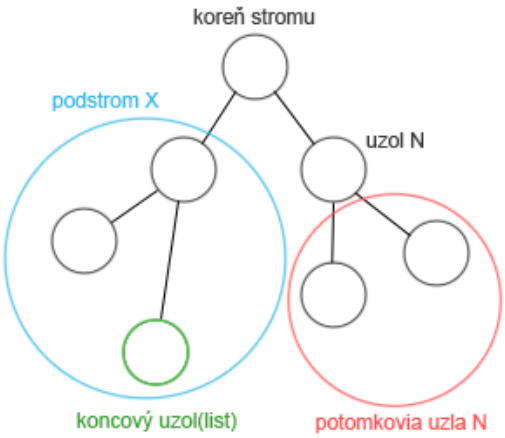
Špeciálny uzol stromu sa nazýva **koreň** (koreňový uzol). Je to najvyšší uzol stromu, ktorý nemá rodiča a všetky d'alšie uzly sú jeho potomkami.

Pod–strom

Pod–strom **X** stromu **Y** je taký strom [8], ktorý sa skladá z uzla stromu **Y** a všetkých jeho potomkov v strome **Y**.

Výška uzla

Výškou uzla [8] označujeme dĺžku najdlhšej zostupnej cesty od daného uzla ku koncovému uzlu (listu).



Obr. 2: Ukážka stromovej štruktúry

Hĺbka uzla

Hĺbkou uzla [8] označujeme dĺžku cesty od daného uzla ku koreňu (koreňovému uzlu).

4.2 Základne typy stromov

Stromové štruktúry môžme rozdeliť na niekoľko základných skupín z hľadiska ich využitia:

- **Binárne stromy**
  - *Binárny strom, Binárny vyhľadávací strom, Red-black strom*
- **B-stromy**
  - *B–strom, B+ strom, B- strom*
- **Multi-way stromy**
  - *Ternárny strom*
- **Stromy rozdeľujúce priestor**
  - *Kvadrantový strom*

V následujúcich riadkoch bude uvedený bližší popis binárneho vyhľadávacieho stromu, B–stromu a kvadrantového stromu.

Binárny vyhľadávací strom

Všeobecne je binárny strom definovaný podľa definície 4.1 [13]

Definice 4.1 *Binárny strom je štruktúra rekurzívne definovaná nad konečnou množinou uzlov, ktorá buď neobsahuje žiadny uzol alebo je zložená z troch disjunktných množín uzlov: koreňa, jeho ľavého a pravého pod–stromu.*

Binárny vyhľadávací strom alebo taktiež „usporiadaný binárny strom“ je špeciálny typ binárneho stromu kde každý uzol obsahuje kľúč na ktorého doméne je definované nejaké usporiadanie. Uzly sú vždy vkladané do stromu podľa následujúcej definície 4.2 [11, 12, 13]:

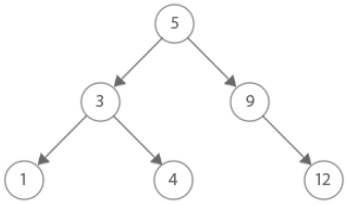
Definice 4.2 *Nech x je uzol binárneho vyhľadávacieho stromu. Ak y je uzol v jeho ľavom pod–strome potom y.key $\leq$ x.key. Ak y je uzol v jeho pravom pod–strome potom y.key $\geq$ x.key*

Na obrázku 3 je ukážka jednoduchého binárneho vyhľadávacieho stromu, kde:

- *root.key = 5*
- *uzly jeho ľavého pod-stromu(1,3,4) $\leq$ root.key*
- *uzly jeho pravého pod-stromu(9,12) $\geq$ root.key*
- *Uzly 1,4,12 na obrázku sú listové uzly zatiaľ čo ostatné sú vnútorné uzly*

Binárny vyhľadávací strom je rýchly čo sa týka vkladania a vyhľadávania. Priemerná zložitosť algoritmu vyhľadávania v strome s počtom uzlov n je $O(\log n)$ [12]. Preto binárne vyhľadávacie stromy sú dobré pre „dictionary“ problémy kde sa vkladá a vyhľadáva informácia indexovaná nejakým kľúčom.

Zložitosť $O(\log n)$ je priemerný prípad. Môže sa líšiť v závislosti na tvare konkrétneho stromu [12]. V najhoršom prípade je to $O(n)$.



Obr. 3: Ukážka binárného vyhľadávacieho stromu

B–strom

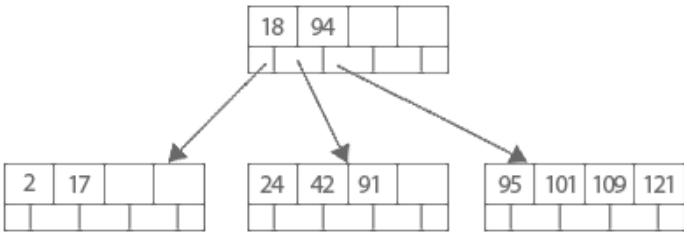
B-strom (viz. obrázok 4) je to vyvážená perzistentná dátová štruktúra, ktorá v uzloch obsahuje c až $2 * c$ položiek. Je široko používaná v databázových systémoch ako indexovacia štruktúra. V prípade B–stromu uzly nazývame stránkami [13].

Definice 4.3 *B-stromu [13]. B-strom rádu c je* $(2*c+1)$*-árny strom, ktorý spĺňuje tieto kritéria:*

- *Každá stránka(uzol) obsahuje najviac* $(2 * c)$ *položiek(kľúčov)*
- *Každá stránka(okrem koreňa) obsahuje aspoň c položiek*
- *Každá stránka je buď listovou(nemá žiadnych nasledovníkov) alebo má m + 1 nasledovníkov kde m je počet kľúčov na stránke*
- *Všetky listové stránky sú na rovnakej úrovni*

Zložitosť základných operácií je $O(\log_c n)$ kde n je počet položiek stromu. Dôležitým faktorom je faktor využitia pamäte tzv. *utilization*, ktorý sa počíta ako (počet položiek uzlu)$*100$. V prípade B–stromu je teda zaručený faktor využitia pamäte minimálne 50%. Z vyššie uvedených vlastností je zrejmé, že B-strom má všetky vlastnosti, ktoré požadujeme po indexovacej dátovej štruktúre [13]:

- *Poskytuje dobrú zložitosť pre základné operácie*
- *Je prirodzene perzistentný*
- *Relatívne ľahko implementovateľný*



Obr. 4: B-strom rádu 2

Kvadrantový strom

Jedná sa o dátovú štruktúru, ktorá bola pomenovaná ako „kvadrantový strom“ R. Finkelom a J.L. Bentleyim v roku 1974 [14].

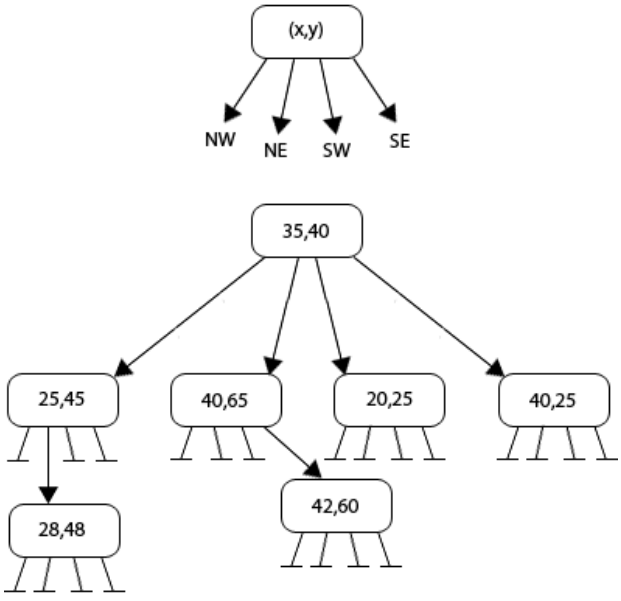
Kvadrantový strom je stromová dátová štruktúra najčastejšie použitá na rozdelenie dvojrozmerného priestoru jeho rekurzívnym delením do štyroch kvadrantov (regiónov). Tie môžu byť v tvare štvorca, obdĺžnika alebo ľubovoľného tvaru [14].

Kvadrantový strom môže byť klasifikovaný vzhľadom na typ dát, ktoré reprezentuje: oblasti, body, linky alebo krivky ale taktiež podľa toho či tvar stromu je nezávislý na poradí v akom sú dáta spracovávané [14, 15]. Medzi bežné typy stromov zaraďujeme *Region Quad tree*, *Point Quad tree* a *Polygonal Map tree*.

Z vyššie uvedených dôvodov je vhodné použiť kvadrantový strom na perzistenciu adresných dat.

Aplikácia používa *bodový kvadrantový strom* viz. obrázok 5, ktorý je adaptáciou binárneho stromu a je používaný na reprezentáciu dvojrozmerných bodových dat. Tvar stromu závisí na poradí v akom sú dáta spracovávané. Je veľmi efektívny z hľadiska porovnávania (hľadania) dvojrozmerných zoradených dátových bodov zvyčajne so zložitosťou $O(\log n)$ kde n je počet bodov v strome.

Každý uzol má práve 4 deti. Zvyčajne sa označujú ako: *SouthWest*, *NorthWest*, *SouthEast* a *NorthEast*. Každý takýto uzol reprezentuje jeden kvadrant [14, 15].



Obr. 5: Ukážka bodového kvadrantového stromu

5 Analýza a návrh

Predchádzajúce dve kapitoly obsahovali základné informácie o predvolenom formáte, ktorý uchováva adresné dáta exportované z projektu OpenStreetMap a o stromových štruktúrach s ktorými bude aplikácia pracovať. Táto kapitola sa bude zaoberať analýzou požiadavkov na aplikáciu na základe ktorých bude aplikácia implementovaná.

5.1 Dátový formát pre uloženie adresných dat

Najdôležitejšou častou návrhu aplikácie bolo zvoliť alebo vytvoriť vhodný dátový formát pre uloženie adresných dát, ktorý zároveň umožní rýchle vyhľadávanie v databáze týchto bodov. Keďže sa jedná o mobilnú aplikáciu, ktorá má omedzenú pamäť bolo nutné zvoliť taký dátový formát, ktorý dosahuje požadovanú výkonnosť z hľadiska vyhľadávania dat v ňom obsiahnutých a ktorý nenadobúda obrovských veľkostí.

Ako existujúci formát pre perzistenciu dat bola zvolená `SQLite` databáza systému Android a vytvorený formát reprezentovaný stromovou štruktúrou realizovanou kvadrantovým stromom. Výstupom obidvoch formátov budú binárne súbory takže výsledná veľkosť bude podstatne menšia oproti pôvodnej. Z dôvodu, že zdrojové dáta exportované z projektu OpenStreetMap dosahujú veľmi obrovských veľkostí[1] bolo nutné vytvoriť vlastný konverzný nástroj pre desktop, ktorý dáta spracuje a skonvertuje do požadovaných formátov. Takto pripravené dáta budú uložené na pamäťové médium mobilného zariadenia a pripravené na použitie.

5.2 Konverzný nástroj

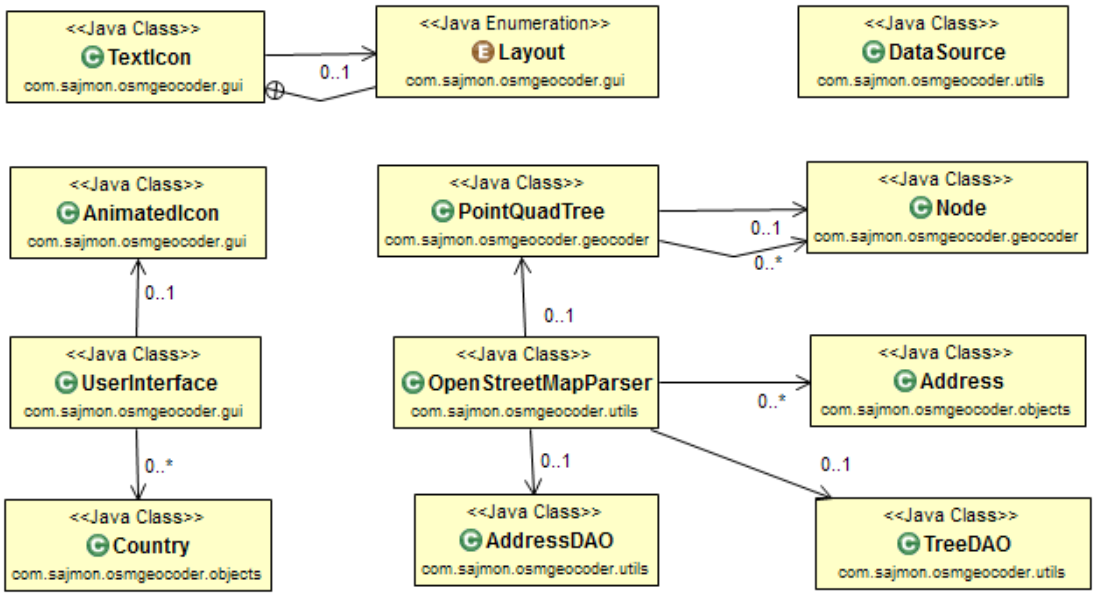
Ako bolo spomenuté vyššie, bolo nutné predvolený textový formát adresných dat skonvertovať do formátu, ktorý bude aplikácia používať tzv. do `SQLite` databázy a stromovej štruktúry. K tomuto účelu bol vytvorený konverzný nástroj pre export adresných dát. Je implementovaný rovnako ako mobilná aplikácia v programovacom jazyku Java (Java SE). Jedná sa o klasickú „javovskú“ aplikáciu s užívateľským rozhraním (ale je možné použiť len exportovanú knižnicu).

Nástroj zdrojové dáta spracuje pomocou knižnice `Osmosis`[2] určenej na „parsovanie“ `OSM` dat. Výstupom sú vygenerované súbory v daných formátoch pripravené na použitie pre mobilnú aplikáciu.

Detailnejší popis nástroja je uvedený v užívateľskej príručke, ktorá sa nachádza v prílohe. Na obrázku 6 je uvedený výsledný triedny diagram konverzného nástroja.

[1] Adresné dáta pre Českú republiku dosahujú okolo cca +- 1 GB

[2] Viz. http://wiki.openstreetmap.org/wiki/Osmosis



Obr. 6: Triedny diagram konverzného nástroja

5.3 Mobilná aplikácia a vzhľad

5.3.1 Užívateľské rozhranie

Pri návrhu aplikácie je návrh užívateľského rozhrania rovnako dôležitý ako návrh aplikačnej logiky, pretože užívateľské rozhranie (ďalej používané ako **UI**) predstavuje „prostriedok“ komunikácie užívateľa s aplikáciou. **UI** je prvá vec s ktorou sa užívateľ oboznámi pri prvom spustení aplikácie a preto by sa mu malo venovať patričnej pozornosti. Je všeobecné známe, že aplikácia môže mať množstvo funkcií ale pokiaľ je rozmiestnenie ovládacích prvkov nelogické a rušivé, väčšinu funkcií užívateľ nikdy nepoužije alebo prinajhoršom ani nebude vedieť , že aplikácia danou funkciou disponuje.

Z vyššie uvedených dôvodov je zrejmé, že ovládacie prvky by mali byť rozmiestnené logický, intuitívne, farby, veľkosti a typ písma by malo byť volené tak aby užívateľa zaujali ale zároveň nepôsobili rušivo čo by mohlo donútiť daného užívateľa o zmene aplikácie, ktorá ponúka rovnakú alebo podobnú funkčnosť.
Práca s **UI** v systéme Android je pohodlná a relatívne jednoduchá z niekoľkých dôvodov:

- *Široká paleta ovládacích prvkov*
- *Možnosť upravovať ovládacie prvky tzv. widgety podľa potreby*
- *Možnosť definovať vlastné tvary, veľkosti a farby*

Kompletné navrhnuté užívateľské rozhranie aplikácie sa nachádza v prílohe.

5.3.2 Vyhľadávacia logika a aplikácia

Inými slovami skrátene **geokódovanie**. Vyhľadávaciu logiku môžme považovať za najdôležitejšiu časť aplikácie takže z tohto dôvodu boli na ňu kladené vysoké požiadavky. Hlavným cieľom bolo navrhnúť požadované triedy a väzby medzi nimi tak aby bola logika aplikácie oddelená od vzhľadu - užívateľského rozhrania (**UI**).

Táto technika má niekoľko výhod:

- *Zdrojový kód je viac čitateľný, riešenie je čistejšie*
- *Jednoduchšia modifikácia aplikácie napr. pridávanie nových funkcií*
- *Vyššia platformová nezávislosť*

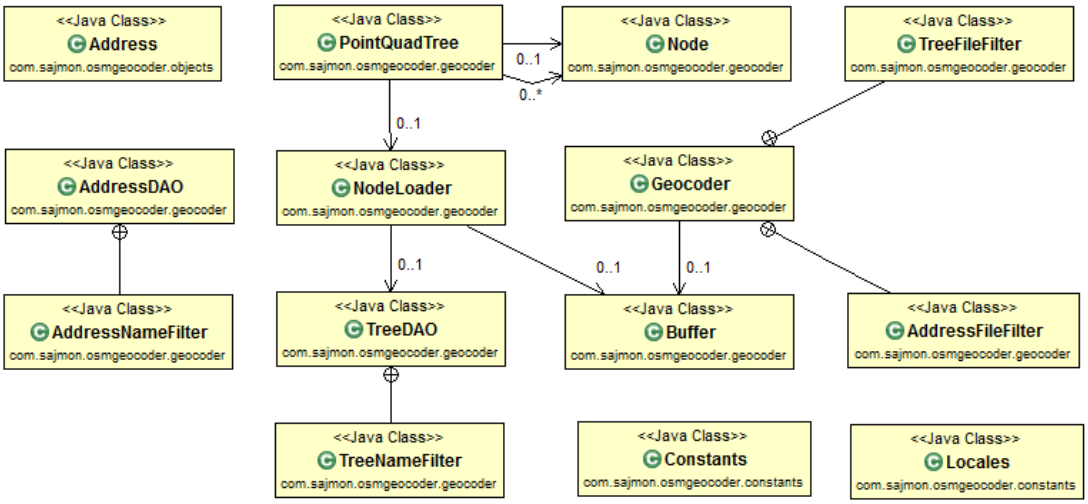
Výsledná knižnica tried bude ponúkať alternatívnu implementáciu k referenčnému API **android.location.Geocoder** o ktorom bude na konci tejto kapitoly uvedená základná špecifikácia a rozdiely oproti vytváranému.

Ďalej bolo nutné sa zamyslieť nad tým či pre bezproblémový a efektívny beh aplikácie bude stačiť keď aplikácia bude bežať práve v jednom vlákne. Teoreticky by to bolo možné (v najideálnejšom prípade) ale z hľadiska efektivity a doporučeným postupom sa vyhľadávacia časť a niektoré ďalšie časti aplikácie (ako napríklad kontrola nainštalovaných máp, inicializácia dátového zdroja) presunuli do vlákna na pozadí tzv. `background Thread`, aby sa zabezpečil bezproblémový chod užívateľského rozhrania.

Na začiatku analýzy bola vyhľadávacia logika presunutá do vlákna na pozadí pomocou triedy, ktorá dedila z vlákna tzv. z `Thread`. Neskôr sa však pri implementácii zistilo, že by bolo vhodné oboznámiť užívateľa informáciou o „aktuálnom stave vyhľadávania“ a zobraziť mu nejakú správu pop. obrázok indikujúci, že „práve prebieha vyhľadávanie“.

Užívateľské rozhranie v systéme Android môže byť modifikované len z originálneho vlákna alebo taktiež všeobecne zaužívaný názov pre toto vlákno je `Main(UI) Thread` takže z tohto dôvodu bolo treba použiť komplexnejší nástroj, ktorý umožňuje ako vykonávanie práce na pozadí tak aj aktualizáciu užívateľského rozhrania a Android nám pre dosiahnutie tohto cieľu ponúka triedu `AsyncTask`. Skrátene je to generická trieda, ktorá vykonáva špecifikovanú prácu asynchrónne a ponúka metódy, ktoré sú zosynchronizované z **UI** vláknom a umožňujú tak aktualizáciu **UI** kedykoľvek je to potrebné. Ďalšou výhodou `generics` je napríklad typová kontrola a bezpečnosť, efektivita.

Na obrázku 7 je zobrazený triedny diagram knižnice, ktorá realizuje požadovanú vyhľadávaciu funkčnosť.



Obr. 7: Triedny diagram knižnice Geocoder

5.4 Referenčné API Geocoder

V tejto sekcii uvediem základné informácie o danom referenčnom API, ktoré ponúka podobnú funkčnosť avšak nástroje niesu úplne totožné.

Jedná sa o triedu, ktorá sa stará o geokódovanie a spätné geokódovanie.

Geokódovanie je proces transformovania adresy alebo popisu miesta adresy do súradníc tzv. *zemepisnej dĺžky a šírky* [5].

Spätné geokódovanie je proces opačný - transformácia súradníc na adresu. Výsledky spätného geokódovania sa môžu líšiť. Jeden výsledok môže obsahovať úplnú adresu pričom druhý môže obsahovať len názov mesta a poštové číslo alebo iné informácie [5].

Trieda pre svoju funkčnosť vyžaduje podpornú službu tzv. internet teda je závislá na pripojení k internetu. Pokiaľ neni internet k dispozícii, výsledkom je 0 nájdených výsledkov a zároveň chyba indikujúca, že služba není dostupná tzv. `IOException` [5].

Trieda implementuje tri metódy používané pri geokódovaní [5]:

- `getFromLocation(double lat, double lon, int maxResults)`

  používa sa prevažne pri spätnom kódovaní teda užívateľ zadá súradnice a metóda mu vráti najlepšie výsledky.

- `getFromLocationName(String locationName, int maxResults)`

  metóda použitá pri geokódovaní. Užívateľ zadá adresu napr. názov ulice a jej číslo a metóda vráti špecifické informácie napr. súradnice adresy, mesto atď.

- `getFromLocationName(String locationName, int maxResults, double lowerLeftLatitude, double lowerLeftLongitude, double upperRightLatitude, double upperRightLongitude)`

  V podstate alternatívna metóda k predchádzajúcej stým rozdielom, že sa užívateľ naviac špecifikuje „hranice“ adresy tzv. pokiaľ vie približné súradnice hľadanej adresy alebo vie napríklad len názov ulice a chce rozsah výsledkov zredukovať.

5.4.1 Rozdiely

Aplikácia, ktorá je predmetom tejto práce ponúka v podstate podobnú funkčnosť ako vyššie spomenuté referenčné API avšak s niekoľkým hlavnými rozdielmi:

- Aplikácia **není závislá** na podpornej službe tzv. na pripojení k sieti Internet
- Spracované zdroje adresných dat sú umiestnené na priamo v mobilnom zariadení teda umožňujú 100% prácu **offline**.
- Vlastná príprava a inštalácia adresných dat pomocou konverzného nástroja
- Možnosť špecifikácie cesty k súborom obsahujúcich spracované adresné dáta prostredníctvom konštruktora knižnice

6 Implementácia

Implementácia aplikácie prebiehala podľa analýzy v predchádzajúcej kapitole. Najskôr bolo potrebné spracovať adresné dáta exportované z *open-source* projektu OpenStreetMap do formátu, ktorý umožní ich rýchle vyhľadávanie a ktorý bude používať mobilné zariadenie. Na vyhľadávanie podľa zadanej adresy (napr. názvu ulice) tzv. **geokódovanie** bol použitý formát reprezentovaný `SQLite` databázou a na vyhľadávanie podľa adresných súradníc tzv. **reverzné geokódovanie** bol použitý formát reprezentovaný kvadrantovým stromom. Na transformáciu adresných dat do požadovaných formátov bola použitá vyššie zmienená konverzná aplikácia.

Keď bola táto časť hotová, boli implementované metódy potrebné pre vyhľadávanie v adresných dátach, ktoré boli postupne upravované a vylepšované pre dosiahnutie čo najlepších výsledkov. Z počiatku bola kontrola výsledkov realizovaná špecifickými výpismi pomocou logcatu a veľmi jednoduchým **UI**. Až vo finálnej fáze sa začalo dotvárať finálne **UI**.

6.1 Zdrojový kód

Od začiatku bol kladený dôraz nato, aby bola logika aplikácie oddelená od jej vzhľadu s cieľom zvýšiť funkčnosť a čitateľnosť kódu pričom pri dodržiavaní tohto nepísaného pravidla sa taktiež kód stáva oveľa ľahšie modifikovateľný. Taktiež bol zdrojový kód rozdelený do menších logických celkov, ktoré teraz stručne a jednoducho opíšem.

Knižnica

Zdrojový kód knižnice bol rozdelený do troch následujúcich balíčkov:

- `com.sajmon.osmgeocoder.geocoder`
- `com.sajmon.osmgeocoder.objects`
- `com.sajmon.osmgeocoder.constants`

Najdôležitejším balíčkom knižnice je prvý balíček `geocoder`, ktorý zapúzdruje hlavnú logiku teda obstaráva vyhľadávanie adries. Je to sada tried, ktorá ponúka alternatívnu implementáciu k `android.location.Geocoder` s niekoľkými rozdielmi spomenutými v predchádzajúcej kapitole.

Obsahuje napríklad hlavnú triedu `Geocoder`, ktorá implementuje potrebné metódy pre vyhľadávanie a ktorá spolupracuje s ďalšími triedami ako napríklad s `DAO` triedami, ktoré obstarávajú spojenie s dátovým zdrojom.

Balíček `com.sajmon.osmgeocoder.objects` obsahuje objekt `Address` používaný knižnicou.

Posledným balíčkom je `com.sajmon.osmgeocoder.constants`, ktorý obsahuje napríklad triedu `Constants`, ktorá zaobstaráva verejné statické premenné používané knižnicou ako napr. názvy stĺpcov v tabuľkách.

Mobilná aplikácia

- `com.sajmon.bc.sampleapp.ui`
- `com.sajmon.bc.sampleapp.async`
- `com.sajmon.bc.sampleapp.db`
- `com.sajmon.bc.sampleapp.adapters`
- `com.sajmon.bc.sampleapp.objects`
- `com.sajmon.bc.sampleapp.holders`
- `com.sajmon.bc.sampleapp.constants`
- `com.sajmon.bc.sampleapp.io`

Prvý balíček tzv. `package` obsahuje triedy užívateľského rozhrania. Môžme povedať, že predstavuje „prezentačnú vrstvu“ mobilnej aplikácie. Obsahuje celkovo 8 aktivít (obrazoviek) medzi ktoré patrí napríklad aktivita zobrazujúca výsledky hľadania alebo uložené adresy.

Balíček `com.sajmon.bc.sampleapp.async` obsahuje asynchronných „workerov“, ktorí sú používaní napríklad pri vyhľadávaní adries, kontrole nainštalovaných máp atď.

Balíček `com.sajmon.bc.sampleapp.db` obsahuje triedy, ktorá slúžia na obsluhu internej databázy aplikácie, do ktorej si môže užívateľ ukladať adresy ak to bude z nejakého dôvodu potrebovať.

Balíček `com.sajmon.bc.sampleapp.adapters` obsahuje adaptéry používané pri zobrazovaní výsledkov (vyhľadaných adries) alebo nainštalovaných máp a pod.

Balíček `com.sajmon.bc.sampleapp.objects` obsahuje objekty používané v aplikácii ako napríklad objekt `Country`, `Map` alebo `Record`.

Balíček `sampleapp.holders` obsahuje napríklad triedu `AddressHolder`, ktorá v úlohe arbitrárneho objektu uchováva v sebe „child views“. Holder je v systéme Android **návrhový vzor**, ktorý slúži na „recyklovanie views“ a je zvyčajne používaný v kombinácii s adaptérovými widgetmi.

Balíček `com.sajmon.bc.sampleapp.io` obsahuje triedu `ExtStorageManager` pre manipuláciu s externou pamäťou (napríklad testovanie prítomnosti SD karty).

Posledným balíčkom je `com.sajmon.bc.sampleapp.constants`, ktorý obsahuje triedu `SQLConstants` ktorá uchováva napríklad názvy stĺpcov internej databázy ako verejné statické premenné.

6.2 SQLite databáza

Ako prvý formát na perzistenciu adresných dáta bola vybraná `SQLite` databáza systému Android.

6.2.1 Schéma

Na obrázku 8 sa nachádza databázová schéma použitá pre mobilnú aplikáciu. Databáza obsahuje jednu tabuľku s názvom `Address`, ktorej názov je samo vysvetľujúci - reprezentuje konkrétnu adresu. Obsahuje 11 stĺpcov, ktoré predstavujú jednotlivé atribúty adresy ako napr. ulica, číslo ulice, mesto atď. Na stĺpce, ktoré sa najčastejšie používali pri selekcii boli vytvorené **indexy** za účelom zvýšenia výkonu vykonávaných dotazov. Každá adresa je jednoznačne identifikovaná umelým **primárnym kľúčom**, ktorý reprezentuje prvý stĺpec s názvom **id**.

V aplikačnej vrstve bola vytvorená objektová reprezentácia danej tabuľky kde vlastnosti objektu tzv. „properties“ sú ekvivalentné k stĺpcom v tabuľke.

Ukážka vytvorenia tabuľky a indexov pomocou jazyka SQL:

```
create table if not exists Address(
    id long primary key,
    lat double not null,
    long double not null,
    conscription_number varchar(20),
    house_number varchar(20),
    street varchar(90),
    street_number varchar(20),
    city varchar(20),
    postcode varchar(20),
    country varchar(20),
    placeholder varchar(1)
);

create index if not exists street_idx on Address(street);
create index if not exists streetnumber_idx on Address(street_number);
create index if not exists conscription_number_idx on Address(conscription_number);
create index if not exists city_idx on Address(city);
create index if not exists placeholder_idx on Address(placeholder);
```

Výpis 2: Ukážka vytvorenia tabuľky a indexov

id	lat	long	conscription_number	house_number	street	street_number	city	postcode	country	placeholder
33705330	49.7021197	17.0731786	678	678/1	Mlýnská	1	Litovel	78401	CZ	m
293889021	50.787292	15.0686588	909	909/32	Michelský vrch	32	Liberec	46014	CZ	m
293889028	50.7445106	15.0629176	384	384/8	Kašparova	8	Liberec	46006	CZ	k
293889032	50.7638351	15.0310748	296	296/33	Nová	33	Liberec	46010	CZ	n
296437111	49.8505487	18.296243	618	618/90	Mariánskohorská	90	Přívoz, Ostrava, Moravskoslezský kraj, CZ	71200	CZ	m
296438005	50.3483047	14.4887583	456	456/7	Nerudova	7	Mělník, Středočeský kraj, CZ	27601	CZ	n
296438436	50.3494926	14.4860511	512	512/26	Krombholcova	26	Mělník, Středočeský kraj, CZ	27601	CZ	k
296438488	50.3530414	14.4990357	1898	1898/31	Nebeského	31	Mělník, Středočeský kraj, CZ	27601	CZ	n
296438834	49.2088439	16.4827147	609	609/21	Klobouček	21	Žebětín, Brno, Jihomoravský kraj, CZ	64100	CZ	k
296438839	50.3451884	14.4987828	2360	2360/12	Okružní	12	Mělník, Středočeský kraj, CZ	27601	CZ	o

Obr. 8: Ukážka štruktúry databázy

6.2.2 Dotazy a užívateľský vstup

Ďalšou dôležitou súčasťou vyhľadávania boli dotazy vykonávané nad databázou s cieľom vrátiť užívateľovi najlepší výsledok. Tu bolo najskôr nutné spracovať užívateľský vstup a na základe typu a počtu hodnôt zostaviť korektný dotaz. Ako základnú kostru dotazu môžme považovať nasledujúcu:

```
select * from Address;
```

do ktorej bola pridávaná selekcia v závislosti na užívateľskom vstupe ak:

- vstup obsahoval nejaké číslo alebo naopak neobsahoval
- počet zadaných slov užívateľom

Prakticky je nemožné predpovedať čo zadá užívateľ – môže zadať len názov ulice, názov ulice a číslo ulice a to buď bez špecifikácie mesta alebo s mestom alebo len jednoducho zadá názov mesta.

Zvolil som riešenie **sadou dotazov**, ktoré boli rôznej skladby a aplikovali sa na základe toho či zadal užívateľ číslo alebo nie a zároveň na počte zadaných **parametrov**.

Vyhľadávanie prebiehalo dovtedy pokiaľ sa nepoužili všetky pred-pripravené dotazy avšak v momente kedy dotaz vrátil výsledok, ostatné sa „zahodili“ a vyhľadávanie bolo ukončené a užívateľovi bol zobrazený výsledok akcie. Vo výpise 3 sú zobrazené niektoré používané dotazy.

```
select * from Address where street like <value> and placeholder = <value>;
select * from Address where city like <value>;
select * from Address where street like <value> and street_number = <value> and city like <
    value> and placeholder = <value>;
```

Výpis 3: Ukážka SQL dotazov

Je vhodné spomenúť, že užívateľ môže zadať aj niečo nevhodné alebo potenciálne nebezpečné s cieľom napríklad zhodiť aplikáciu. Najmä z tohto dôvodu boli na dotazy kladené bezpečnostné požiadavky a boli vykonané následujúce akcie:

- Použitie **parametrizovaných dotazov** z cieľom zvýšiť bezpečnosť a čitateľnosť dotazu
- **Vždy** pred vykonaním dotazu bol užívateľský vstup **kontrolovaný** či neobsahuje potenciálne nebezpečné znaky ktoré boli z dotazu odstránené v prípade ak ich dotaz obsahoval.

Nakoniec po vykonanej kontrole bol užívateľský vstup transformovaný do poľa „bind" parametrov, ktoré bolo špecificky vložené do vytvorených dotazov.

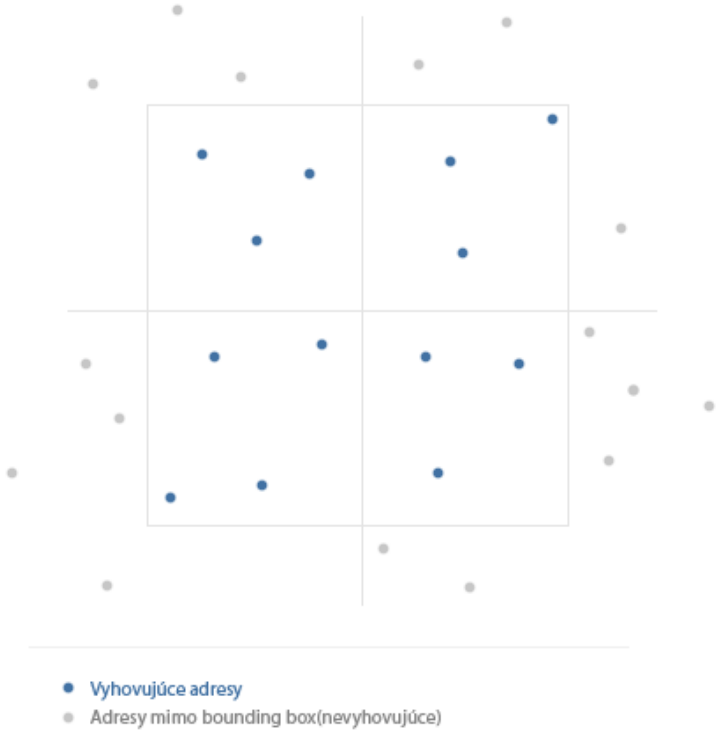
6.3 Kvadrantový strom

Druhým zvoleným formátom bol formát reprezentovaný kvadrantovým stromom, ktorý sa previedol do binárnej podoby a aplikácia následne pri vyhľadávaní adries strom prechádzala a získavala z neho potrebné adresné dáta. Strom sa skladal z kolekcie uzlov, kde každý uzol obsahoval:

- **int id** ako jednoznačný identifikátor uzla v strome
- súradnice teda **zemepisnú šírku a dĺžku**
- **long info** ako jednoznačný identifikátor adresy v `SQLite` databáze.
- jednoznačné identifikátory svojich synovským uzlov `southWest, northWest, southEast, northEast` pričom ak daný identifikátor obsahoval hodnotu **-1** daný uzol (rodič) nemal žiadne dáta priradené k danému synovskému uzlu.

Vyhľadávanie v strome prebiehalo s použitím **rekurzie** a s užívateľom špecifikovaným ohraničujúcim rámčekom tzv. „bounding boxom“ viz obrázok č. 9.

Adresy, ktoré sa nachádzali v „bounding boxe“ boli vrátené ako výsledok dotazu. Ako bolo spomenuté, užívateľ si mohol tento rámček tzv. presnosť vyhľadávania sám nadstaviť (napríklad vyhľadávanie v okruhu 500 metrov). Konečný výsledok je zároveň zoradený od najbližšej nájdenej adresy po najvzdialenejšiu.



Obr. 9: Ukážka bounding boxu

6.4 Problémy pri implementácii

Počas implementácie sa objavilo niekoľko problémov, ktoré z počiatku neboli zrejmé a bolo ich nutné istým spôsobom vyriešiť. Následne stručne popíšem tie „najdôležitejšie“ týkajúce sa hlavnej funkčnosti aplikácie.

6.4.1 Užívateľský vstup

Prvý problém, ktorý stručne zhrniem je užívateľský vstup. Tu nastávali problémy týkajúce sa funkčnosti vyhľadávania ale aj samotnej aplikácie. Pokiaľ sa užívateľ pri zadávaní vstupu (ktorý predstavoval čiastočnú adresu) pomýlil napr. zadal nejaký potenciálne nebezpečný znak, medzery naviac atď. alebo sa dokonca úmyselne snažil napísať nebezpečný kód bolo nutné pred spustením vyhľadávania tento vstup vhodne ošetriť a naformátovať.

K dosiahnutiu tohto cieľa bolo nutné vykonať pár akcií:

- Všetky potenciálne nebezpečné znaky, ktoré vstup obsahoval bolo potrebné odstrániť. Medzi tieto znaky patria napríklad: `[* # % )( & @ $ ; +]`
- Odstránenie nadbytočných medzier tzv. zaistiť, aby bola medzi slovami **práve jedna** medzera
- Používať výhradne **parametrizované dotazy** vykonávané nad databázou

6.4.2 Perzistencia kvadrantového stromu

Ďalším problémom bol spôsob **perzistencie** stromu. Na začiatku implementácie bola myšlienka udržiavať si v pamäti telefónu celý strom a tým pádom mať prístup k všetkým dátam, ktoré obsahoval.

Ak by sa jednalo o klasickú desktopovú aplikáciu tak tento spôsob by bolo možné použiť. Avšak v našom prípade sa jedná o mobilnú aplikáciu kde mobilné zariadenie má výrazne obmedzenejšiu, menšiu pamäť ako desktop kde každý `byte` pamäte hrá veľkú rolu. Zároveň môžme povedať, že výsledkom akéhokoľvek dotazu vykonaného nad stromom vrátane rozsahového je určitý pod-strom `X` stromu `Y` takže nie je nutné si držať v pamäti celý strom (v našom prípade to ani nie je možné). Riešením bola **semi-perzistenia** stromu.

Strom sa pred-pripravil na desktope pomocou konverzného nástroja, ktorý vytvoril jeho binárnu reprezentáciu ktorou bola tabuľka v databáze. Keďže strom je v podstate kolekcia uzlov je možné strom prezentovať tabuľkou v databáze. V našom prípade databáza obsahovala dva stĺpce:

- stĺpec **id** typu `Integer`, ktorý predstavoval **primárny kľúč** a zároveň jednoznačný identifikátor riadku aj uzlu
- stĺpec **node** typu `Blob`, ktorý reprezentoval **serializovaný** uzol

Výstupom bol **binárny súbor**, ktorý sa umiestnil na pamäťové médium a aplikácia ho používala pri vyhľadávaní adries na základe zadaných súradníc teda na reverzné geokódovanie.

Vždy sa zo súboru (databázy) čítali len uzly, ktoré boli výsledkom dotazu a priebežne sa uchovávali vo vytvorenom **buffery**.

Keďže uzly boli uchovávané v buffery bolo potrebné vytvoriť nejakú štruktúru, ktorá sa bude starať o ich načítavanie. Z tohto dôvodu bola pri vyhľadávaní v strome vytvorená pomocná štruktúra pomenovaná ako `NodeLoader`, ktorá sa starala o načítavanie uzlov z buffera.

Pokiaľ uzol, ktorý bolo nutné pri dotaze navštíviť a porovnať jeho hodnoty s hľadanými sa nenachádzal v buffery, vyššie zmienená štruktúra sa postarala o to, že konkrétny uzol načítala z databázy a uložila do buffera pre ďalšie použitie.

Nakoniec v prípade ak buffer obsahoval istý *kritický* počet uzlov čo by mohlo znížiť výkonnosť aplikácie bol buffer vyčistený za účelom zachovania optimálneho behu aplikácie.

7 Testy a dosiahnuté výsledky

Od samotného začiatku vývoja aplikácie bola aplikácia testovaná a ladená na reálnom mobilnom zariadení Samsung Galaxy S Advance s verziou Androidu **2.3.6 Gingerbeard**. Keďže funkčnosť aplikácie vyžadovala neustále čítanie dat z externého uložiska v našom prípade z *SD karty*, použitie emulátora sa nezdalo byť veľmi efektívne. Okrem iného aplikácia taktiež vyžadovala istý výkon kvôli náročnejším operáciám kde rovnako ako v predchádzajúcom prípade emulátor nebol vyhovujúci keďže má podstatne odlišný výkon ako reálne mobilné zariadenie. Emulátor bol využívaný iba pri vytváraní a následnom ladení **UI**.

Potom čo aplikácia dosiahla finálnej fázy bola testovaná na dvoch mobilných zariadenia, ktoré sú uvedené v tabuľke 1.

Mobilný telefón	**Verzia systému**	**Procesor**	**RAM**	**Displej**
Samsung Galaxy S Advance	Android 2.3.6	1 GHz	768 MB	480 x 800 b.
HTC Desire X	Android 4.0.4	1 GHz	768 MB	480 x 800 b.

Tabuľka 1: Testovacie mobilné telefóny

Ako môžme vidieť v tabuľke, aplikácia bola otestovaná na dvoch odlišných verziách Androidu. V oboch prípadoch aplikácia fungovala podľa očakávaní. Zároveň funkčnosť a výsledky aplikácie boli porovnané s výsledkami a funkčnosťou referenčného API **android.location.Geocoder**.

7.1 Porovnanie funkčnosti a výsledkov

Ako bolo spomenuté vyššie, aplikácia bola testovaná a porovnávaná s referenčným API **android.location.Geocoder**. Bola testovaná rýchlosť a presnosť vyhľadávania. Výsledky testov sa nachádzajú v tabuľkách 2 a 3.

Knižnica	**com.sajmon.Geocoder**	**android.location.Geocoder**	
Pripojenie	Žiadne	Wi-Fi	WCDMA
Najkratší čas hľadania	62 ms	102 ms	324 ms
Najdlhší čas hľadania	7857 ms	2435 ms	4912 ms
Priemerný čas hľadania	2063.25 ms	454.25 ms	2023.625 ms
Úspešnosť	90%	95 %	95 %

Tabuľka 2: Výsledky testov pre klasické geokódovanie

Knižnica	**com.sajmon.Geocoder**	**android.location.Geocoder**	
Pripojenie	Žiadne	Wi-Fi	WCDMA
Najkratší čas hľadania	100 ms	102 ms	262 ms
Najdlhší čas hľadania	1536 ms	670 ms	4517 ms
Priemerný čas hľadania	506.75 ms	221.5 ms	1907.25 ms
Úspešnosť	95%	100%	100%

Tabuľka 3: Výsledky testov pre reverzné geokódovanie

Kde „Najdlhší čas hľadania“ v tabuľke 2 nastal v oboch prípadoch ak adresa nebola úspešné nájdená.

Z výsledkov je zrejmé, že referenčné API bolo na tom lepšie najmä z hľadiska rýchlosti a to hlavne pri **klasickom geokódovaní**.

Je nutné ale zohľadniť fakt, že aplikácia, ktorá je predmetom tejto práce sa odlišuje od referenčného API tým, že ponúka prácu **offline** a všetky výpočty a ďalšie nutné úkony sa vykonávajú priamo v telefóne pričom u referenčného rozhrania prebieha všetko online na výkonných serveroch s ktorými sa výkon mobilnej aplikácie nemôže porovnávať.

Testovalo sa na základe 20 vyhľadaných adries v štandardnom tvare a to pre pre oba prípady teda pre klasické aj reverzné geokódovanie. Aplikácia obstála výborne, výsledky boli ekvivalentné s výsledkami referenčného API. Je vhodné spomenúť, že u reverzného kódovania sa výsledky trocha odlišovali. Jednalo sa len o niekoľko desiatok až stoviek metrov čo môže byť spôsobené odlišnou metrikou výpočtu vzdialenosti dvoch súradníc alebo určovania najlepších resp. najbližších adries.

8 Záver

Cieľom práce bolo podať užívateľovi základné a utriedené informácie o operačnom systéme Android, ktorý sa v súčasnej dobe teší vysokej popularite a to aj vďaka možnostiam, ktoré ponúka v oblasti vývoja aplikácií a detailne popísať vývoj a implementáciu knižnice, ktorá ponúka alternatívnu implementáciu k existujúcemu API

android.location.Gecoder a aplikácie, ktorá demonštruje jej funkčnosť.

Na začiatku práce som uviedol základné informácie o operačnom systéme Android, jeho histórii, architektúre a možnostiach vývoja pre danú platformu. Následne som podal základné informácie o open-source projekte OpenStreetMap, ktorý poskytuje zdrojové dáta ktoré aplikácia používa a stromových štruktúrach, ktoré aplikácia rovnako používa k zaisteniu svojej funkcionality.

Zbytok práce sa venuje výhradne samotnej aplikácii, popisuje ako bola aplikácia navrhnutá, jej funkcie a nakoniec popisujem jej implementáciu a výsledky vykonaných testov a funkčnosti.

Čo sa týka samotnej aplikácie a jej výhľadov do budúcnosti. Aplikácia dosahuje dobré výsledky ale určite sa nájdu veci, ktoré by sa dali zlepšiť. Za zmienku stojí spomenúť použitie rôznych heuristických metód, ktoré z nedostatku času neboli aplikované a ktoré zaručia kompresiu zdrojových dat keďže aktuálne dáta môžu dosahovať niekoľko stoviek MB čo nie je málo keďže sa jedná o mobilnú aplikáciu s omedzenejšou kapacitou pamäte a výkonu oproti klasickému desktopu. Možné by bolo taktiež rozšírenie o routing (navigácia, výpočet tras).

Tak isto by sa dala samozrejme ešte viac optimalizovať rýchlosť a kvalita vyhľadávania pri klasickom ako aj reverznom geokódovaní ako napríklad použiť lepší algoritmus pri vyhľadávaní najbližších adries, napr. `K-NN`.

Taktiež stojí za úvahu popremýšľať o možnosti rozšíriť aplikáciu aj medzi ďalšie konkurenčné platformy akými sú iOS od spoločnosti Apple a Windows Phone od spoločnosti Microsoft.

9 Literatúra

[1] WEI-MENG, Lee. *Beginning Android 4 Application Development, version: March, 2012* Wrox Press, ISBN: 978-1-1181-9954-1

[2] Svět Androida: Vyvíjíme pro Android – úvod. [online]. [cit. 2013-04-29] *Dostupné z* <http://www.svetandroida.cz/vyvijime-pro-android-1-uvod-201103>

[3] OSMZ: Android. [online]. [cit. 2013-04-29] *Dostupné z* <http://osmz.wikidot.com/android>

[4] SQLite: About SQLite. [online]. [cit. 2013-04-29] *Dostupné z* <http://www.sqlite.org/about.html>

[5] Android: Android developers. [online]. [cit. 2013-04-29] *Dostupné z* <http://developer.android.com/index.html>

[6] OpenStreetMap Wiki: Main Page. [online]. [cit. 2013-04-29] *Dostupné z* <http://wiki.openstreetmap.org/wiki/Main_Page>

[7] EHRINGER, David. The Dalvik Virtual Machine Architecture, March, 2010. [online]. [cit. 2013-04-29]

Dostupné z <http://show.docjava.com/posterous/file/2012/12/10222640-The_Dalvik_Virtual_Machine.pdf>

[8] IdeaInfo: Trees. [online]. [cit. 2013-04-29]

Dostupné z <http://ideainfo.8m.com/>

[9] I-Programmer: Data structures - Trees. [online]. [cit. 2013-04-29]

Dostupné z <http://www.i-programmer.info/babbages-bag/477-trees.html>

[10] KNUTH, Donald: The Art of Computer Programming: Fundamental Algorithms, Third Edition

Addison-Wesley, 1997. ISBN 0-201-89683-4

[11] CORMEN Thomas, LEISERSON Charles, RIVEST Ronald, STEIN Clifford: Introduction to Algorithms, Second Edition

Hardcover, ISBN: 0072970545

[12] PARLANTE, Nick. Binary Trees. [online]. [cit. 2013-04-29]
Dostupné z <http://cslibrary.stanford.edu/110/BinaryTrees.pdf>

[13] KRÁTKY, Michal a Radim BAČA. Databázové systémy. [online]. [cit. 2013-04-29]
Dostupné z <http://dbedu.cs.vsb.cz/SubPages/OpenFile/OpenFile.aspx?file=dbcb/dbcb.pdf>

[14] FINKEL Raphael, BENTLEY J.L, Quad Trees: A Data Structure for Retrieval on Composite Keys
Acta Informatica, Volume 4

[15] DE BERG Mark, VAN KREFELD M., OVERMARS M., SCHWARZKOPF O., Computational Geometry: Algorithms and Applications, 2nd edition
Springer, ISBN: 3540656200

A Práca s konverznou aplikáciou

Ako bolo spomenuté v dokumente vyššie, zdrojové dáta exportované z *open-source* projektu OpenStreetMap:

- Dosahovali veľmi obrovských veľkostí, niekedy aj viac ako 1 GB
- Formát nepodporoval rýchle vyhľadávanie dat obsiahnutých v ňom

Hlavne z týchto dôvodov vznikla konverzná aplikácia, ktorej úlohou je zdrojové dáta spracovať a skonvertovať do formátov používaných aplikáciou. Následne sú tieto predspracované dáta nahrané na pamäťové médium mobilného zariadenia a sú prakticky pripravené na použitie. Predtým ako si povieme ako s aplikáciou pracovať, uvediem základné informácie o tom, ako aplikácia funguje a aké konvencie používa:

- Aplikácia ako vstup podporuje len zdrojový OSM textový formát, ktorý predstavuje primárny formát dat exportovaných z projektu OpenStreetMap. Pokiaľ by boli dáta v inom formáte, napríklad v `PBF` formáte, je možné použiť program **osmconvert**[3] ktorý dáta skonvertuje do OSM formátu.
- Výstupom budú minimálne 2 binárne súbory reprezentujúce databázu dat a kvadrantový strom reprezentovaný taktiež tabuľkou v databáze (aplikáciu je potrebné spustiť ako administrátor aby nedošlo k problémom pri vytváraní súborov).
- Tieto súbory budú v tvare:
  - `[locale]db.db` pre databázu adries
  - `[locale]db[order].db` pre databázu adries v prípade ak sa vytvára viac ako jeden databázový súbor[4]
  - `[locale]tree.db` pre kvadrantový strom

Ukážka výstupu programu sa nachádza na obrázku 10.

[3] Je k dispozícii na oficiálnych stránkach projektu OpenStreetMap
[4] Vysvetlenie sa bude nachádzať v podkapitole: Ako s aplikáciou pracovať

Obr. 10: Ukážka výstupu konverzného nástroja

A.1 Ako s aplikáciou pracovať

Na obrázku 11 sa nachádza úplné užívateľské rozhranie konverznej aplikácie. Cieľom bolo poskytnúť jednoduché, intuitívne **UI**, ktoré razantne uľahčí prácu potenciálnemu užívateľovi a neodradí ho od jeho použitia napríklad z dôvodov zložitej obsluhy. Na obrázku môžete vidieť následujúce prvky:

1. **ComboBox 1:**

   Prostredníctvom ktorého sa vyberie krajina tzv. `locale`, ktorej adresné dáta budú skonvertované a následne použite v mobile ako „nainštalovaná mapa“.

2. **ComboBox 2:**

   Jedná sa o ďalší **ComboBox** pomocou ktorého sa vyberie počet databázových súborov, ktoré sa vytvoria. Toto nastavenie je veľmi dôležité použiť ak sa jedná o väčšiu krajinu, ktoré bude mať pravdepodobne veľké množstvo adresných dat za účelom optimalizovať rýchlosť vyhľadávania v týchto dátach. Rozdelenie prebehne na základe znaku, ktorým sa adresa začína.

   Následne bude vždy vygenerovaný jeden extra databázový súbor do ktorého sa uložia všetky špeciálne prípady adries:

   (a) *Ak adresa začína číslom*

   (b) *Ak začína nejakým špeciálnym znakom, ktorý sa nenachádza v tabuľke ASCII*

3. **CheckBox**

   Ak je potrebné vytvoriť **práve jeden** databázový súbor (platí pre menšie krajiny), „zaškrtnutý stav“ znamená, že výstupom bude práve jeden databázový súbor bez ďalšieho rozdelenia.

4. **InputBox 1**

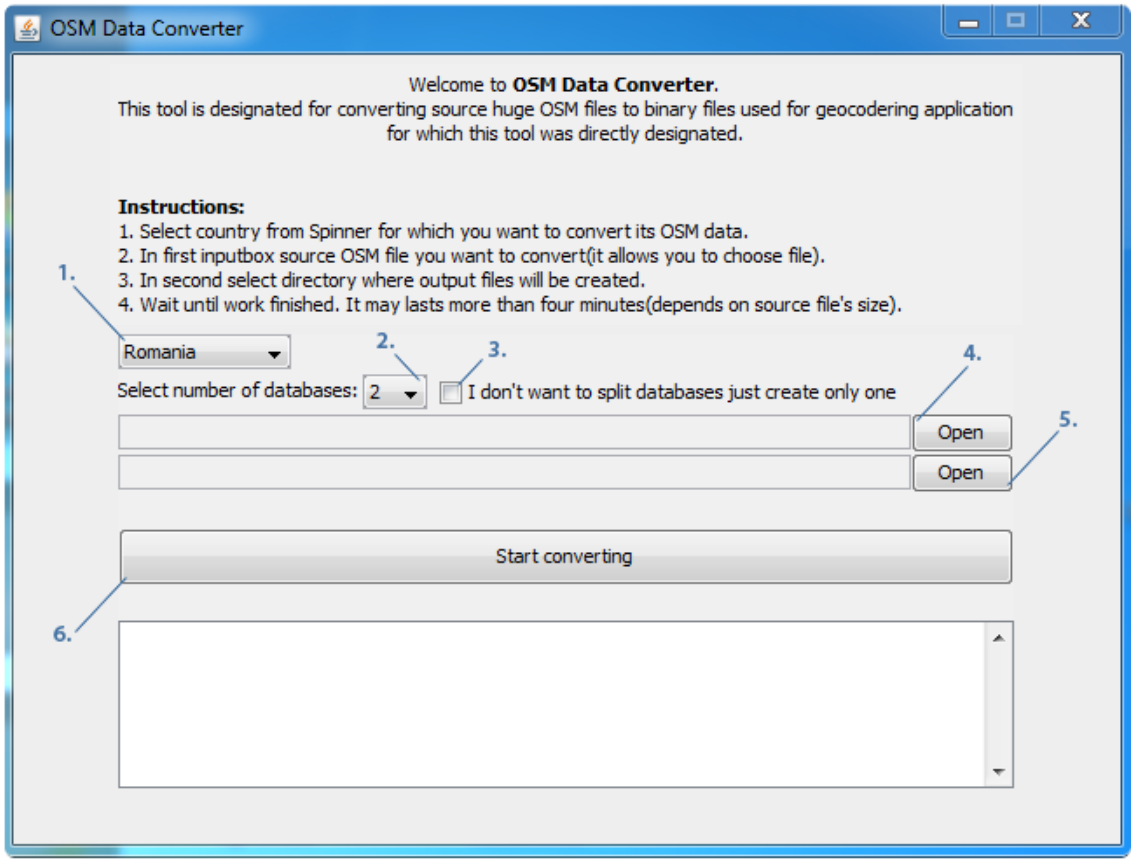
   Do daného poľa bude vložená cesta k zdrojovému súboru, ktorý má byť skonvertovaný. Je vhodné pripomenúť, že jediný podporovaný formát je `[name].osm`

5. **InputBox 2**

   Tu platí to isté ako v predchádzajúcom prípade avšak s tým rozdielom, že sem bude vložený adresár (zložka) do ktorej nástroj uloží vygenerované a skonvertované súbory.

6. **Button**

   Už z názvu tlačítka je zrejmé aku má funkciu. Týmto tlačítkom sa spustí samotné konvertovanie, ktoré môže trvať aj niekoľko minút v závislosti na rýchlosti vášho počítača a veľkosti zdrojových dat.



Obr. 11: Ukážka užívateľského rozhrania konverzného nástroja

B Inštalácia mobilnej aplikácie

V úvode je nutné poznamenať, že aplikácia nepochádza z Google Play takže nato aby bolo možné aplikáciu nainštalovať do mobilného zariadenia je nutné mať povolené „inštalácie z iných zdrojov“.

Aplikáciu je možné nainštalovať viacerými spôsobmi:

- Skopírovať aplikáciu (súbor `.apk`) do zariadenia, vybrať ju pomocou súborového manažéra a nainštalovať cez inštalátor balíčkov
- Nahrať `.apk` súbor na nejaký filehosting (napríklad na dropbox) a po stiahnutí súboru OS Android automaticky ponúkne možnosť inštalácie aplikácie
- Pripojiť mobilné zariadenie k desktopu pomocou USB kábla a využiť nástroj `adb` s následujúcim príkazom [5]

  ```
  adb -s [sériové číslo zariadenia] install [cesta k .apk súboru]
  ```

Kde parameter `-s` špecifikujúc sériové číslo zariadenia je nepovinný a je ho nutné zadať len v prípade ak je pripojených viac zariadení aby nástroj vedel do ktorého zariadenia má aplikáciu nahrať a nainštalovať. Ak je pripojené len jedno zariadenie, príkaz je implicitne zaslaný do daného zariadenia.

Sériové číslo zariadenia je možné zistiť príkazom [5] `adb devices`

B.1 Inštalácia máp

Ide o súbory exportované konverzným nástrojom opísaným v sekcii 38. Pri inštalácii máp platí pravidlo, že súbory exportované z konverzného nástroja musia byť uložené v zložke, ktorej názov je ekvivalentný s „locale“ súboru tzv:

```
[locale]db.db
```

Napríklad ak súbory majú „locale“ `cz` tak zložka musí mať ekvivalentný názov **cz**. Pri použití mobilnej aplikácie sa takto vytvorené „mapy“ vkladajú do vstavanej zložky aplikácie vytvorenej na SD karte `com.sajmon.geocoder.data/databases`

Pri použití knižnice musia byť vytvorené zložky vložené v nejakej rodičovskej zložke, ktorá môže byť ľubovoľne pomenovaná[5]. Táto zložka sa potom spolu s „locale“ predáva do konštruktora triedy `Geocoder`, ktorú obsahuje knižnica.

[5] Zvyčajne ako root, databases alebo maps

C Inštalácia knižnice

Nato, aby bolo možné knižnicu použiť v nejakom inom projekte, je nutné vložiť `.jar` súbor do „Build Path“ projektu. To sa dosiahne tým, že sa najskôr musí vytvoriť zložka **libs** v koreňovom priečinku projektu viz. obrázok 12. Následne sa do tohto priečinka presunie `.jar` súbor knižnice.

Potom už len stačí kliknúť pravým tlačítkom myšky na súbor a zvoliť „Add to Build Path“ viz. obrázok 14 čím by sa mali vytvoriť „Referenced libraries“ viz. obrázok 13 a inštalácia knižnice je hotová.

Pokiaľ by nebola knižnica vložená do Build Path, pri snahe spustiť aplikáciu by najpravdepodobnejšie došlo k chybe `NoClassDefFound` alebo `ClassNotFound`.

C.1 Overenie funkčnosti knižnice

Po vložení `.jar` súboru do zložky **libs** a vloženia knižnice do Build Path projektu by mala byť knižnica pripravená na použitie.

Za predpokladu správneho nainštalovania knižnice by mal byť pri jej použití prítomný následujúci import:

```
import com.sajmon.osmgeocoder.geocoder.Geocoder;
```

C.2 Príklad použitia knižnice

V následujúcom výpise 4 sa nachádza základné použitie knižnice:

```
String rootPath = Environment.getExternalStorageDirectory().getAbsolutePath();
rootPath += "<path>/rootTestDir";
String value = params[0];
List<Address> list = new ArrayList<Address>();
Geocoder g = new Geocoder(c, rootPath, "cz", 500);
list = g.getFromLocationName(value, 10);
if (! list.isEmpty()) {
    for (Address a: list) {
        Log.i("Information", a.getStreet() + " " + a.getStreetnumber() + ", " + a.getCity());
    }
}
```

Výpis 4: 1. ukážka použitia knižnice

Výstupom[6] ukážkového kódu výpisu 4 bude 10 vyhľadaných adries.

[6]Výstup bol zvolený logcat pre účel demonštrácie funkčnosti knižnice

Stručné vysvetlenie kódu:

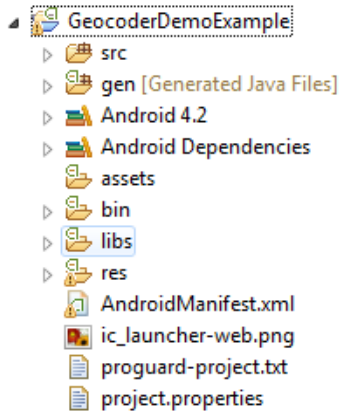
- **rootPath:** je zložka na SD karte obsahujúca nainštalované mapy tzv. zložky (cz, sk, en atď.) kde `<path>` parameter je nepovinný pretože u každého zariadenia sa môže líšiť resp. ho zariadenie nemusí ani mať.
- **value:** sú užívateľom zadané vstupné dáta (v danom príde čiastočná adresa ako String)
- **list:** prázdny list adries, ktorý bude naplnený adresami vrátenými knižnicou Geocoder
- **g:** je vytvorená inštancia knižnice preberajúca parametre:
  - **Context:** kontext[7] aktivity do ktorej je Geocoder pripojený
  - **String:** „root“ zložka obsahujúca nainštalované mapy
  - **String:** špecifikovaný „locale“ určujúci pre ktorý štát sa bude vyhľadávať
  - **double:** špecifikovaný bounding box (presnosť vyhľadávania) pri reverznom geokódovaní

Uvedený príklad je ukážkou klasického geokódovania kde sa zadáva čiastočná adresa. Opakom je reverzné geokódovanie pri ktorom sa zadávajú približné súradnice adresy. K tomu je nutné použiť metódu `getFromLocation` viz. výpis 5:

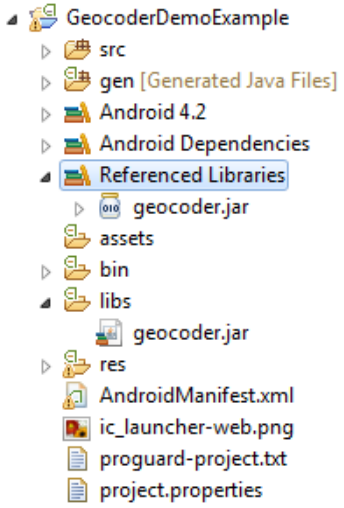
```
double lat = 49.835;
double lon = 18.165;
Geocoder g = new Geocoder(c, rootPath, "cz", 500);
 list  = g.getFromLocation(lat, lon, 10);
if (! list .isEmpty()) {
   for (Address a: list ) {
      Log.i("Information", a.getStreet() + "␣" + a.getStreetnumber() + ",␣" + a.getCity());
   }
}
```

Výpis 5: 2. ukážka použitia knižnice

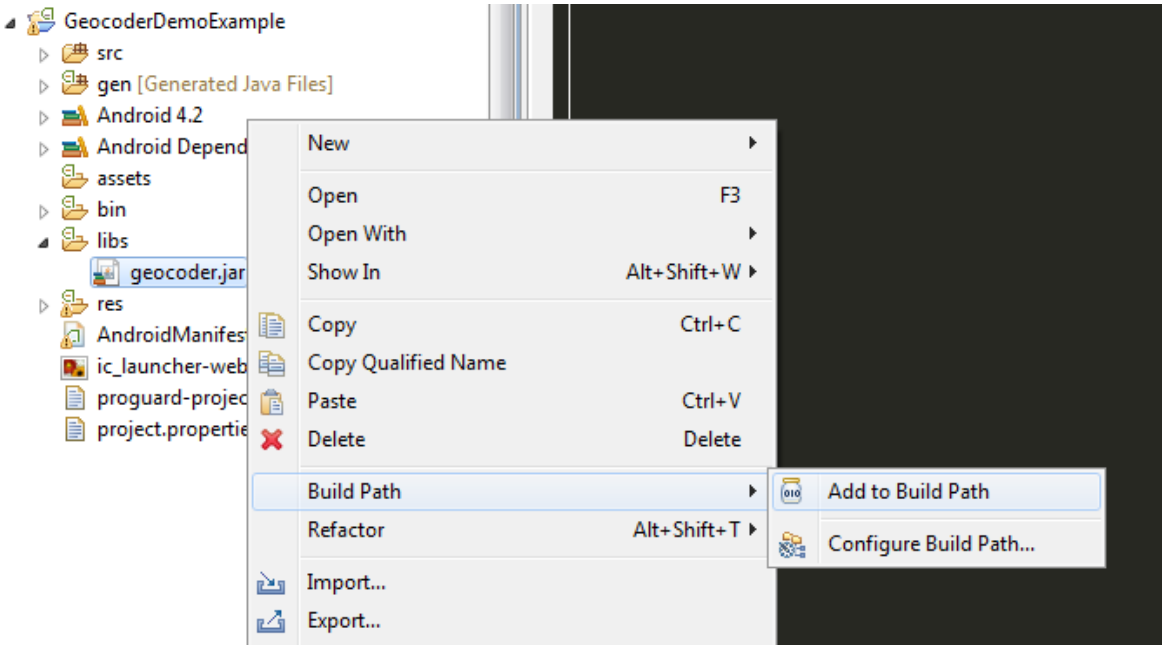
[7] Detaily na http://developer.android.com/reference/packages.html



Obr. 12: Vytvorenie zložky libs



Obr. 13: Referenced libraries



Obr. 14: Vloženie knižnice do Build Path

D Popis mobilnej aplikácie

Pokiaľ bola aplikácia úspešne nainštalovaná na mobilné zariadenie, môže byť spustená. Následne je zobrazená úvodná obrazovka aplikácie viz. obrázok 15. Aby bolo možné začať vyhľadávať adresy je nutné zvoliť štát resp. „mapu“ v ktorej bude prebiehať vyhľadávanie viz. obrázok 16. Keď je zvolený štát, môžu sa začať vyhľadávať adresy. K dispozícií sú dva módy vyhľadávania:

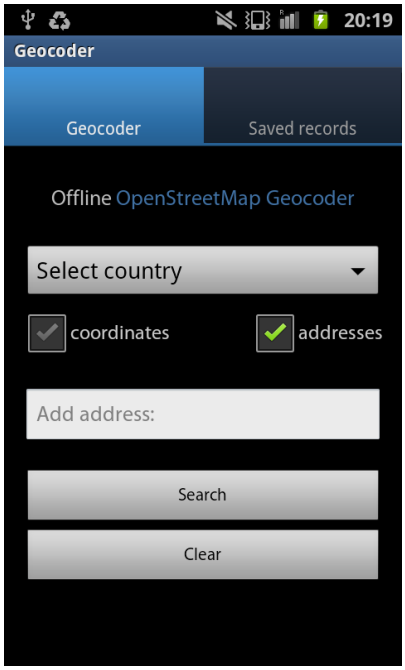
- Klasické geokódovanie pri ktorom zadávame časť adresy do poľa viz. obrázok 17
- Reverzné geokódovanie pri ktorom zadávame približné súradnice hľadanej adresy viz. obrázok 18

Po zadaní vstupných údajov a stlačení vyhľadávacieho tlačítka prebehne verifikácia vstupu. Pokiaľ je vstup platný, začne sa vyhľadávať. V opačnom prípade je užívateľovi zobrazená varovná správa o tom, že nezadal korektné vstupné údaje.

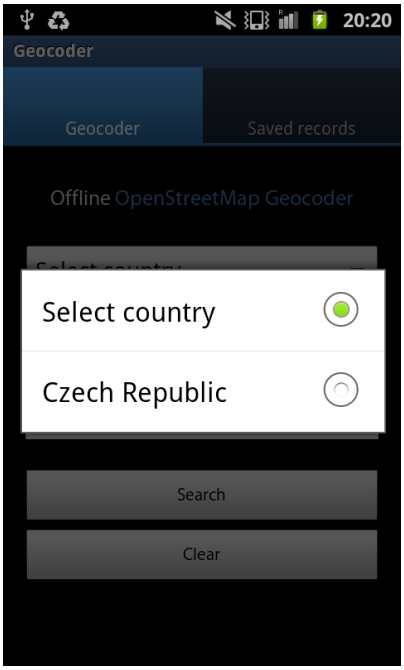
Následne je užívateľ o aktuálnom stave vyhľadávania informovaný prostredníctvom dialógu viz. obrázok 19.

Pokiaľ bolo hľadanie úspešné sú užívateľovi zobrazené výsledky viz. obrázok 20. V tomto stave môže užívateľ vykonať niekoľko akcií prostredníctvom kontextového menu (zobrazenom na obrázku 21):

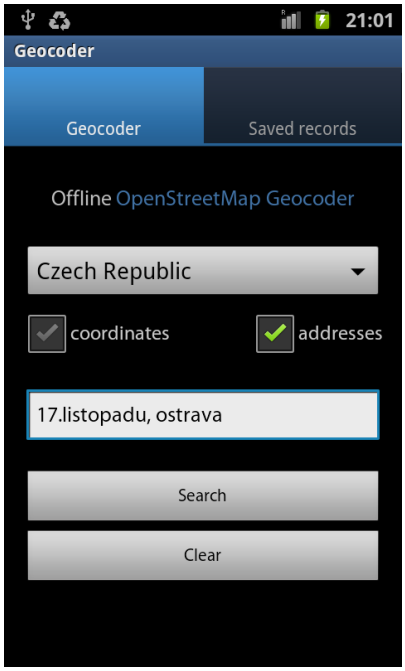
- Zobraziť dodatočné informácie o adrese viz. obrázok 22
- Uložiť adresu a následne zobraziť uložené adresy viz. obrázky 21 a 23
- V prípade, že má telefón aktívne pripojenie k internetu je tu možnosť zobrazenia adresy na mape viz. obrázok 24



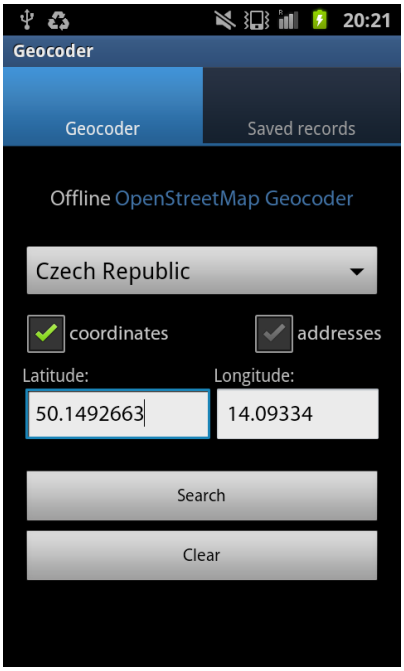
Obr. 15: UI: Úvodná obrazovka



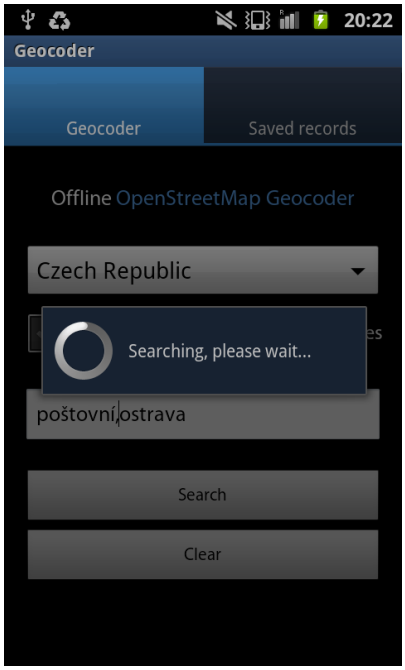
Obr. 16: UI: Selekcia štátu



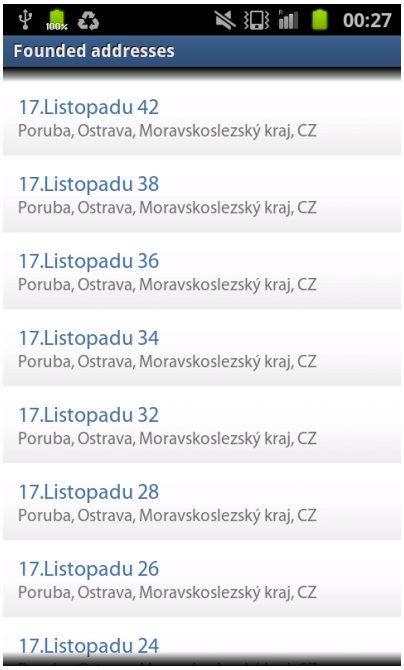
Obr. 17: UI: Klasické geokódovanie



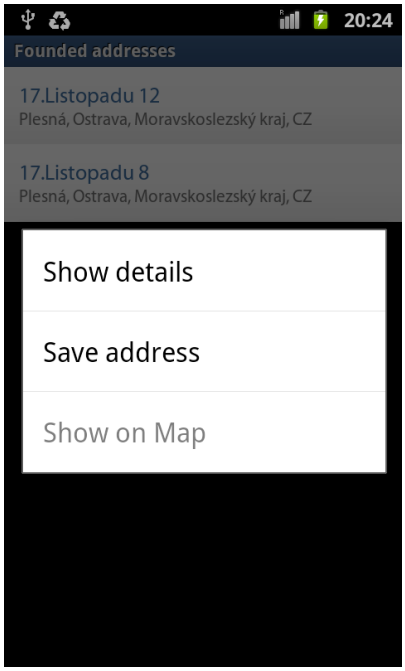
Obr. 18: UI: Reverzné geokódovanie



Obr. 19: UI: Dialóg a stav vyhľadávania

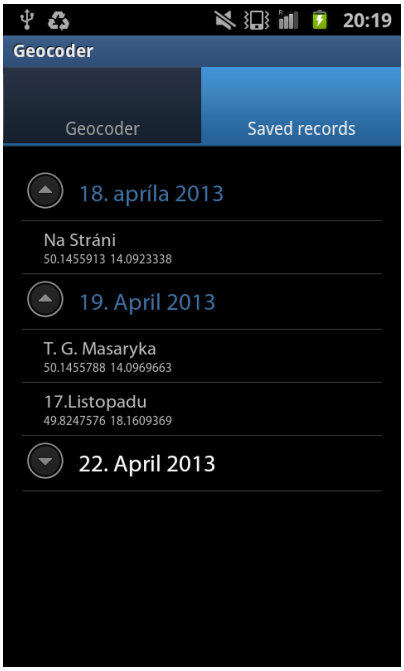


Obr. 20: UI: Výsledky hľadania

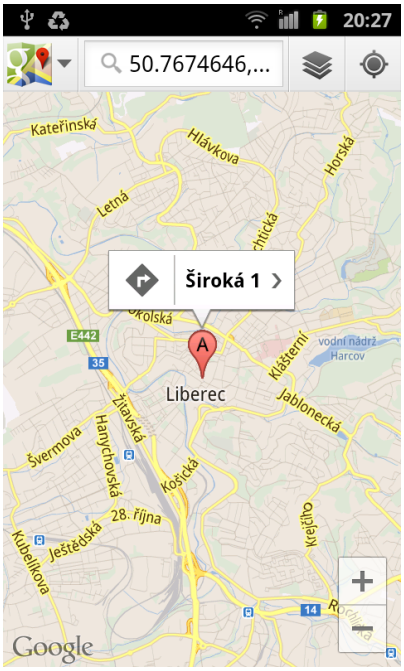


Obr. 21: UI: Kontextové menu

Obr. 22: UI: Detail adresy



Obr. 23: UI: Uložené adresy



Obr. 24: UI: Adresa zobrazená na mape

E Obsah priloženého CD

Na priloženom CD sa nachádza:

- **GeocoderKnižnica** - `.jar` súbor knižnice
- **GeocoderKnižnicaProjekt** - zdrojové kódy knižnice (Eclipse)
- **MobilnáAplikácia** - inštalačný súbor `.apk` mobilnej aplikácie
- **MobilnáAplikáciaProjekt** - zdrojové kódy mobilnej aplikácie (Eclipse)
- **KonverznáAplikácia** - spustiteľný `.jar` súbor
- **KonverznáAplikáciaProjekt** - zdrojové kódy konverznej aplikácie (Eclipse)
- **Dáta** - zdrojové dáta pre Českú republiku
- **Dokument** - text bakalárskej práce vo formáte `.pdf`